Rapport de projet de fin d'études
Bees for life

JAWADI Ahmed - JOLLY Cédric - REGNIES Anthony

université de BORDEAUX

Bees For Life
Lutter contre les frelons asiatiques

janvier - mars 2022

Clients :
ECONOMIDES Marie - DESBARATS Pascal - WILLAERT Lionel

TABLE DES MATIÈRES

Remerciements

Nous remercions l'association *Bees for Life*, ainsi que notre enseignant Monsieur Pascal Desbarats pour avoir proposé ce sujet. Grâce au suivi régulier hebdomadaire avec également Madame Marie Economides, nous avons pu être guidés tout au long du projet. Nous tenons également à remercier Monsieur Lionel Willaert pour les différents dialogues que nous avons eu avec lui pour savoir ce qu'il souhaitait avoir comme ajout et de mieux comprendre son processus d'utilisation. Cela nous a aussi permis de voir le projet dans sa globalité et quelles seront les directions prises après le rendu du projet.

Introduction

Présentations

JAWADI Ahmed, **JOLLY Cédric**, **REGNIES Anthony** sommes étudiants en Master informatique, spécialisé dans le traitement de l'image et du son. C'est dans le cadre de notre Projet de fin d'étude (PFE) que nous avons été amenés à choisir un sujet de travail de programmation pour conclure notre formation.

Parmi les projets proposés, notre choix s'est porté sur celui-ci : "*Bees For Life*", de part notre intérêt collectif pour le traitement d'images par "l'intelligence profonde" mais aussi par l'utilisation des drones pour l'acquisition d'images. C'est enfin pour son aspect environnemental que nous avons choisi ce sujet, et peut-être un peu également pour notre haine des petits insectes.

Ce PFE se déroule sur une période de 2 mois et demi, de mi-janvier à fin mars 2022. Nous avons été encadrés tout au long de ce projet par **M. DESBARATS Pascal**: enseignant chercheur à l'université de Bordeaux, et **Mme ECONOMIDES Marie**, doctorante. Le client final de ce projet était l'association homonyme "*Bees For Life*", représenté par **M. WILLAERT Lionel**.

Le projet

Bees For Life est une application qui vise à lutter contre le déploiement des frelons asiatiques dans le sud de la France lié au réchauffement climatique. L'association du même nom s'emploie à localiser et retirer tous les nids de frelons asiatiques signalés dans la région.L'application Bees For Life a été conçue pour aider à cet objectif : Lorsqu'un frelon asiatique est signalé, l'association envoie un drone survoler tout le secteur afin de localiser le nid. Une fois le vol terminé, les images à la fois thermique et visible sont analysées par l'application. Un algorithme de *deep learning* est utilisé pour détecter automatiquement toute suspicion de nid à la fois sur le visible et le thermique. Un spécialiste n'a alors plus qu'à consulter les emplacements suspects pour confirmer ou non la présence d'un nid. Il peut alors aller sur place grâce aux coordonnées GPS de la photo pour déloger les frelons.

Notre travail consistait à faire évoluer cette application déjà développée lors de précédents PFE. Les objectifs principaux étaient les suivants:

- Faire évoluer l'algorithme de *deep learning* pour le faire fonctionner avec le nouveau drone dont s'est équipée l'association.
- Analyser et afficher la température des nids sur les images thermiques.

- Afficher précisément la position GPS du nid pour faciliter sa localisation
- Améliorer la fiabilité du logiciel actuel
- Tester de nouvelles techniques d'apprentissages et comparer les résultats avec la méthode actuelle.
- Améliorer l'interface graphique de l'application pour faciliter l'exploitation des résultats.

Etat de l'art

De nombreux travaux ont déjà été fait sur le sujet des frelons asiatiques certains plus ou moins éloignés de notre projet. En effet, un papier traitant exactement du sujet dont nous traitons a été écrit par notre client en partie **[1]**. Nous étudierons dans les prochaines parties l'état de l'art portant sur l'identification d'objet sur des images, les 3 principales méthodes étudiées étant *YOLO*, les *Masks-RCNN* et les *Vision Transformers*.

YOLO, la première idée

Définition

You Only Look Once (YOLO) est un algorithme de détection d'objets connu pour sa grande précision et sa rapidité, proposé par Joseph Redmond *en* 2015. Comme son nom l'indique, il traite toute l'image d'un seul coup.

Architecture

YOLO réduit la taille des images d'un facteur 32, appelé la stride du réseau. La première version de *YOLO* prenait des images de taille 448×448, ainsi la feature map de sortie était de dimension 14×14.

Il est courant que les objets que l'on veuille détecter soient au centre de l'image. Or, une grille de taille 14×14 n'a pas de centre unique. En pratique, il est donc préférable que la sortie ait une taille impaire. Pour lever cette ambiguïté, la taille des images sera 416×416, pour fournir une *feature map* de taille 13×13 avec un centre unique.

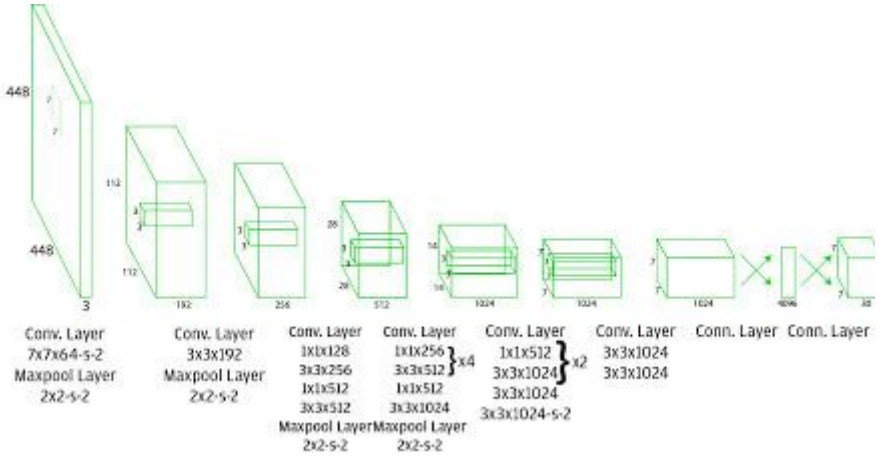

Fig 1: Architecture Yolo

(source : https://www.geeksforgeeks.org/yolo-you-only-look-once-real-time-object-detection)

Pour bien comprendre *YOLO*, il faut comprendre ses sorties [Fig 1]. *YOLOv3* dispose de trois couches finales, la première a une dimension divisée par 31 par rapport à l'image initiale, la deuxième par 16 et la troisième par 8. Ainsi en partant d'une image de taille 416×416 pixels, les trois features maps en sortie du réseau auront des tailles respectives de 13×13, 26×26 et 52×52 pixels. C'est en ce sens que *YOLOv3* prédit trois niveaux de détails, pour détecter respectivement les gros, moyens et petits objets.

Partant d'une image de taille 416×416 pixels, un même pixel est « suivi » à travers le réseau et conduit à trois cellules. Pour chaque cellule trois *bounding box* sont prédites, cela en fait un total de 9 qui sont issues des 9 ancres. Pour chaque *bounding box*, un score *d'objectness* et des scores d'appartenance aux classes sont prédits. Au total, le réseau propose 52×52×3 + 26×26×3 + 13×13×3 = 10647 *bounding boxes*.

YOLO a été le premier algorithme testé sur l'application *Bees for Life* lors du premier PFE. Il n'a cependant pas été retenu, ses résultats n'étant pas compétitifs face à son successeur : les *Masks-RCNN*.

CNN et Masks-RCNN

Les *Masks-RCNN (Region Convolutional Neural Networks)* sont le type de modèle utilisé dans l'application au moment où nous arrivons sur le développement du projet. Le principe de ce réseau de neurones est de prendre une image en entrée et de localiser en sortie des boîtes englobantes avec des labels associés **[2]**.
Le réseau calcule les boîtes englobantes à partir des masques d'entraînement. Pour préserver l'entièreté des images, on applique un *zero-padding* sur les images après qu'elles aient été redimensionnées. Le réseau stocke uniquement les pixels du masque à l'intérieur des boîtes englobantes pour économiser de la mémoire et augmenter la vitesse d'entraînement. La *RPN (Region Proposal Network)* se charge quant à elle de calculer les anchors (repérer les zones d'intérêts). Ensuite, pour une image de test, la *RPN* calcule en utilisant le modèle avec les poids calculés précédemment sur les anchors des images pour donner un score aux objets trouvés (donc les nids). Finalement, le réseau va dire si une région contient un nid ou non et va attribuer une probabilité aux anchors appelées positives. Le masque pourra ensuite être généré à partir des boîtes encombrantes [Fig 2].

Fig 2: Exemple de Masks R-CNN sur une image de nid dans un pin.

VIT pour Vision Transformers

Le *Vision Transformer* est une application récente des Transformers au traitement d'image. A l'origine les transformers étaient surtout utilisés et reconnus pour leur efficacité dans le traitement automatique des langues **[3]**.

Les *Transformers* mesurent les relations entre les paires d'entrées, c'est l'attention. A l'origine surtout utilisé pour les logiciels de traduction automatique, il se concentre sur les liens entre les successions de mots. Pour les images, l'unité d'analyse de base est le pixel. Mais la quantité de pixels par image étant trop importante, le ViT calcule les relations entre les pixels de petits patchs de l'image (par exemple de 32x32 pixels) **[4]**.

Le *ViT* applique donc l'architecture transformée sur ces patchs d'images successifs sans utiliser de couches de convolutions [Fig 3].

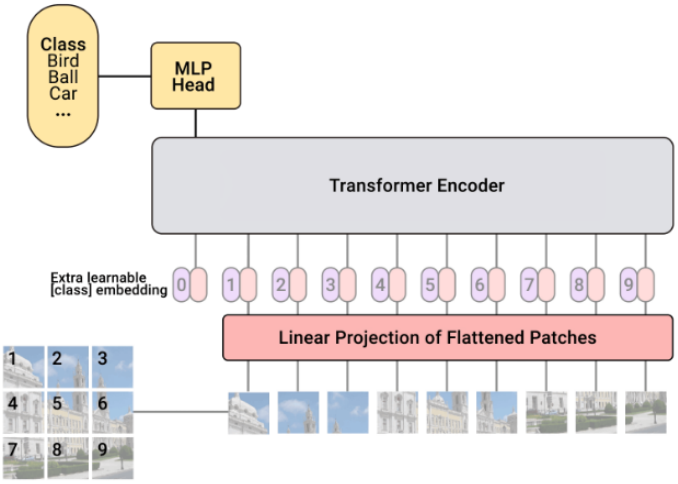

Fig 3: Schéma du processus de l'algorithme des ViT
(source : https://viso.ai/deep-learning/vision-transformer-vit/)

Dans le *ViT*, la création des patchs d'images est considérée comme une couche du réseau de neurones. En entrée l'image complète est donc découpée en patch par la première couche du réseau. [Fig 4]

Fig 4: Visualisation de la division en plusieurs patchs d'une image

Le modèle *ViT* se compose de plusieurs *blocs Transformer*, qui utilisent des couches *"MultiHead Attention"*. Ces blocs produisent un tenseur 3D, qui est traité par un classifieur avec *softmax* pour produire en sortie nos probabilités par classes.
Dans notre cas nous avons 2 classes : 0 pour une image de fond, 1 pour une image qui contient un nid de frelons.

> *La "MultiHeadAttention" est le système qui permet de connecter les patchs entre eux. Il part du principe que les informations présentes sur le patch précédent sont importantes pour comprendre les informations du patch courant. (Concept utilisé à l'origine pour la traduction où lorsque l'on veut traduire une phrase, les autres mots de la phrase ont une importance pour comprendre le contexte du mot courant).*

Après 100 époques, le modèle *ViT* atteint une précision d'environ 55% sur la base de données *CIFAR-100*. Le résultat n'est pas incroyable en soit puisque des réseaux convolutionnels peuvent obtenir 76% avec les mêmes données. Cependant l'intérêt des *ViT* c'est qu'ils semblent efficaces même avec de faibles quantités de données en entrée et en utilisant moins de ressources. C'est de par ailleurs aujourd'hui le principal concurrent au *CNN* et *R-CNN* pour l'identification d'image, il est donc impératif d'étudier ses résultats sur nos images. On peut toutefois s'interroger sur la pertinence de la logique des *Transformers* dans notre cas qui consiste à localiser un nid isolé sur une image : les *ViT* risquent de ne pas trouver de logique à la succession des patchs puisque le nid sera petit et localisé. Enfin un autre danger est de réussir à faire fonctionner le *ViT* ici utilisé sur des petites images de 32x32 pixels, sur nos images HD de 8000x6000 pixels. Un défi de taille qui risque de rendre le *ViT* inexploitable sans trop détériorer l'image au point de la rendre inutilisable.

User Stories

USER STORY 1 :
Le client insère les images visible et thermique du drone dans les dossiers respectifs et lance le code pour trouver les nids de frelons.

USER STORY 2 :
Le client souhaite pouvoir comparer en même temps l'image résultante thermique et visible ainsi que l'image d'origine..

USER STORY 3 :
Le client consulte l'application pour lire la température d'un nid visible et déterminer son niveau d'activité.

USER STORY 4 :
Après avoir lancé la détection des nids, le client souhaite voir sur une carte la position précise des nids détectés.

USER STORY 5 :
Le client souhaite présenter les résultats de l'efficacité du logiciel à ses clients.

USER STORY 6 :
Le client souhaite que le drone puisse détecter en temps réel la position d'un nid et se rapprocher pour l'analyser de plus prêt et confirmer ou infirmer la suspicion.

USER STORY 7 :
Le client souhaite agrandir et zoomer l'image pour voir le nid.

Les Besoins

Besoins fonctionnels

Besoin fonctionnel 1.1.
Analyser les images couleur pour détecter les nids de frelons présents.
Besoin fonctionnel 1.2.
Analyser les images FLIR pour détecter les nids de frelons présents.

Besoin fonctionnel 2.1.
Consulter les résultats de la détection sur les images visible et thermique simultanément.
Besoin fonctionnel 2.2.
Consulter l'image d'origine pour confirmer la présence d'un nid après l'affichage d'un masque RCNN.

Besoin fonctionnel 3.1.
Consulter la position GPS d'une image où un nid a été détecté.
Besoin fonctionnel 3.2.

Consulter sur une carte la position de toutes les images où un nid a été détecté.

Besoin fonctionnel 4.
Présenter avec des chiffres l'efficacité du réseau de l'application pour la détection.
Besoin fonctionnel 5.
Utiliser les images thermique pour connaître l'état d'activité d'un nid.
Besoin fonctionnel 6.
Permettre à d'autres utilisateurs d'envoyer leurs images sur un site pour qu'elles soient traitées par l'application.

Besoins non-fonctionnels

Besoin non-fonctionnel 1. L'analyse des images fournies pour la détection se fait en un temps raisonnable (moins d'une minute par image).
Besoin non-fonctionnel 2. La localisation du nid est suffisamment précise pour permettre de le localiser facilement (rayon de recherche <10m).
Besoin non-fonctionnel 3. L'interface graphique doit être simple d'usage et intuitive pour permettre à n'importe qui de s'en servir.

Il est difficile d'énoncer une liste de besoins non-fonctionnels pour ce projet, voici une brève liste de quelques besoins non-fonctionnels identifiés. Cependant, le cœur du projet est avant tout de répondre à des besoins fonctionnels, les aspects propres au temps de réponse de l'application sont secondaires.

Planning

Pour la gestion du projet, nous nous sommes aidés du logiciel trello permettant de faire un kanban en ligne. Composé de 4 colonnes (*Stories*, *TODO*, *In progress*, *To verify* et *Done*). Nous avons donc placé nos tâches dans ce tableau pour avancer progressivement tout au long du projet et s'assurer que chaque fonctionnalité était testé par un autre membre avant d'être validé. Pour utiliser ce kanban, nous avons élaboré une liste des tâches qui a parfois été amenée à évoluer tout au long du projet en fonction des discussions que nous avons pu avoir avec le client.

Liste des tâches

Besoin fonctionnel 1. Analyse des images :
tâche 1.1. Garantir le bon fonctionnement de la détection sur les images de l'ancien drone.
tâche 1.2. Mettre à jour l'entraînement de l'application pour le faire fonctionner au mieux sur les images du nouveau drone.

Besoin fonctionnel 2. Consulter les résultats.
tâche 2.1. Afficher simultanément les images thermiques et couleurs.
tâche 2.2. Agrandir l'affichage des images pour profiter de la haute qualité de celles-ci.
tâche 2.3. Permettre de cacher / afficher le mask RCNN pour confirmer à l'œil nu la présence d'un nid de frelon.

Besoin fonctionnel 3. Consulter la position GPS d'une image.
tâche 3.1. Récupérer la position GPS du drone pour l'afficher.
tâche 3.2. Générer un QRcode pour pouvoir récupérer la position du nid sur Google Maps.
tâche 3.3. Afficher dans l'application une carte avec la position des nids suspectés et pouvoir filtrer selon la fiabilité de la détection.
tâche 3.4. Calculer la position précise d'un point sur une image.

Besoin fonctionnel 4. Chiffrer l'efficacité des réseaux.
tâche 4.1. Dessiner les graphiques présentant la précision de l'algorithme tout au long l'entraînement.
tâche 4.2. Calculer le taux d'erreur et de réussite avec les différentes valeurs (vrai positif, faux positif…)

Besoin fonctionnel 5. Utiliser les images thermique pour connaître l'état d'activité d'un nid.
tâche 5.1. Analyser l'image thermique pour en extraire la température du nid.

Besoin fonctionnel 6. Site internet
tâche 6.1. Transformer l'application en site internet pour la rendre accessible à tous.

Diagramme de PERT

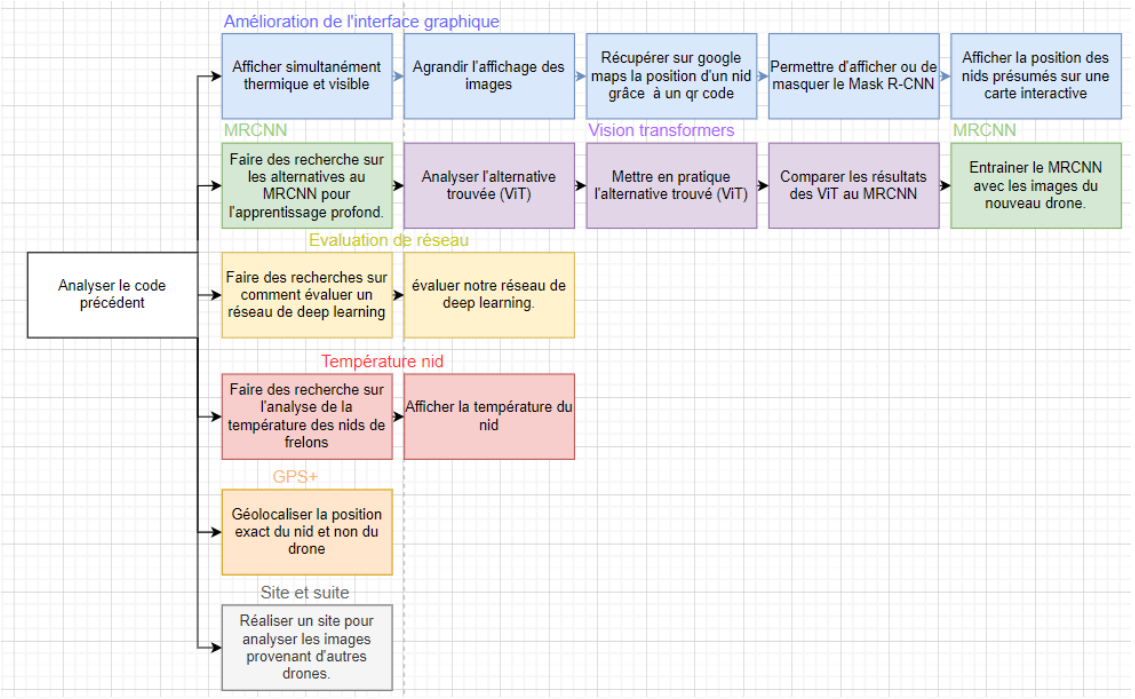

Fig 5: Diagramme de PERT du projet

Diagramme de GANTT

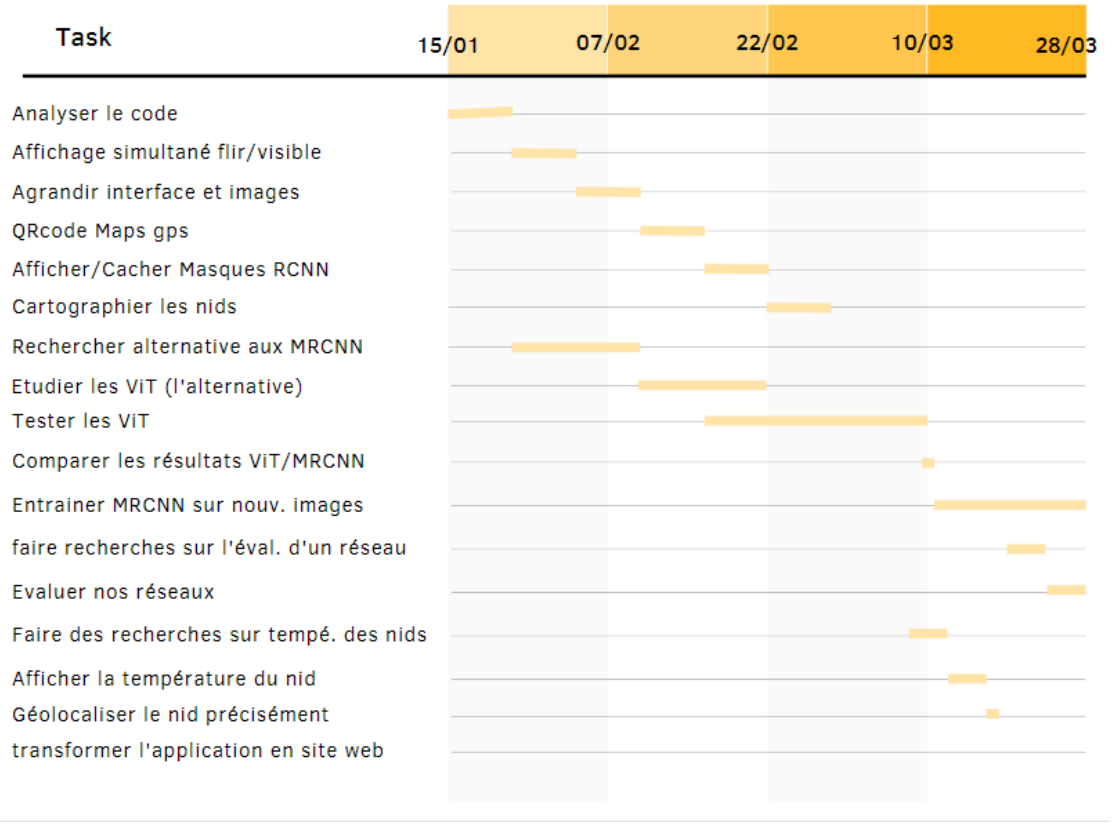

Fig 6: Diagramme de GANTT du projet

Architecture

Choix logiciels

Le projet *Bees for Life* existant déjà avant notre arrivée, le choix logiciel s'est vite imposé, nous avons donc continué le projet en python comme c'était déjà le cas auparavant. Nous avons également choisis de continuer de travailler avec les versions de *tensorflow* et les librairies précédemment utilisées [Fig 7].

Présentation de l’architecture

Fig 7: Présentation de l’architecture

Installation

Le projet se trouve sous la forme d’une archive à décompresser. Si cela n’est pas fait, l’installation de **Python 3** ainsi que de **Anaconda** est nécessaire. Ensuite en ouvrant un terminal et en se dirigeant dans le dossier décompressé du projet (avec la commande **cd** sur linux et mac).

Pour installer et faire fonctionner le logiciel, il est nécessaire d'exécuter le script **install.sh (Linux/Mac)** ou **install.bat (Windows).** Ce script va installer toutes les dépendances nécessaires au fonctionnement du logiciel ainsi que créer les répertoires prévus pour stocker les images traitées.

Utilisation

À l'aide de Anaconda, on aura un environnement spécifique qui nous permettra d'utiliser des versions spécifiques de dépendances sans avoir à rendre incompatibles d'autres potentielles applications sur la machine. Il faut alors si l'environnement n'est pas activé écrire dans le terminal **conda activate beeslife3** pour activer l'environnement. Par la suite, le logiciel se lancera grâce au fichier main.py. Pour le lancer il suffit de se déplacer dans le dossier src à l'aide de la commande **cd src** et d'écrire **python main.py**. Le logiciel devrait normalement se lancer avec une interface graphique.

Le travail effectué

Refonte de l'interface graphique

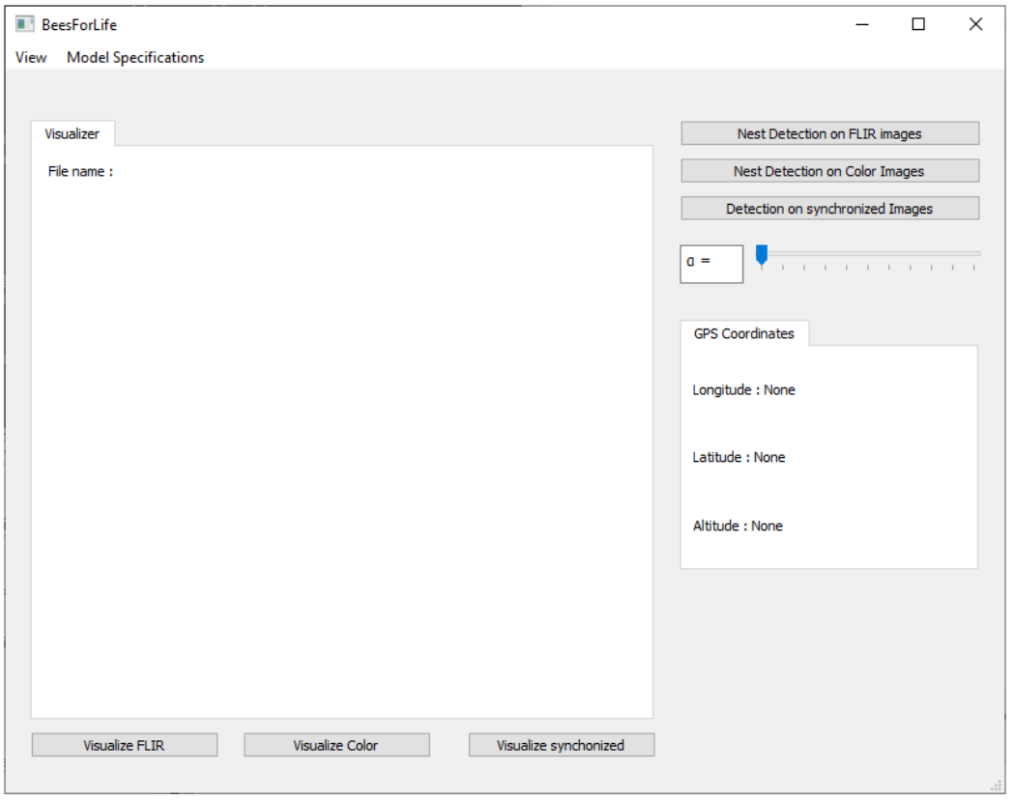

Fig 8: Interface graphique initiale du projet

Lors de notre arrivée sur le projet, l'interface graphique était plutôt sommaire [Fig 8]. De nombreuses requêtes du client portaient sur l'amélioration de cette interface pour faciliter l'utilisation et améliorer l'exploitation des résultats. En ce sens, plusieurs tâches portaient sur des changements de cette interface graphique :

tâche 2.1. Afficher simultanément les images thermiques et couleurs.
tâche 2.2. Agrandir l'affichage des images pour profiter de la haute qualité de celles-ci.
tâche 2.3. Permettre de cacher / afficher le mask RCNN pour confirmer à l'œil nu la

présence d'un nid de frelon.
tâche 3.3. Afficher dans l'application une carte avec la position des nids suspectés et pouvoir filtrer selon la fiabilité de la détection.

Pour répondre à ces tâches nous avons commencé par re-organiser le code et rendre l'application "responsive" pour qu'elle puisse être redimensionnée et qu'on puisse ainsi observer les images en plein écran pour trouver les nids plus facilement.

Également, le précédent logiciel ne permettait pas de parcourir les images une fois les nids repérés. Il a donc fallu ajouter un système pour naviguer entre les images efficacement, tout en affichant en simultané l'image visible et l'image thermique correspondant afin de vérifier plus facilement l'authenticité du nid. Nous avons opté pour un système de slider qui semblait propice à la navigation chronologique des images [Fig 9].

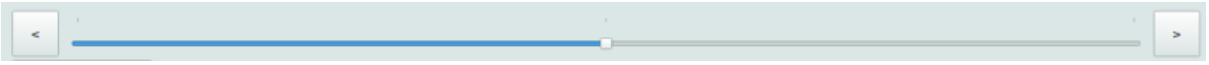

Fig 9: Présentation du slider d'images

Un autre problème repéré par le client après usage du logiciel porte sur la confirmation de la présence d'un nid. Après le passage du *Masks-RCNN,* l'image finale est difficilement visible, il faut donc retourner voir l'image originale pour que l'utilisateur puisse confirmer ou non la présence d'un nid. Nous avons donc ajouté un simple bouton "*SHOW/HIDE MASK*" pour rendre la tâche plus rapide à l'utilisateur. L'image s'affiche également en plus grand puisqu'elle n'est plus détériorée par les différents "0 *padding*" effectués pour la détection (responsable des grosses bordures blanches) [Fig 10].

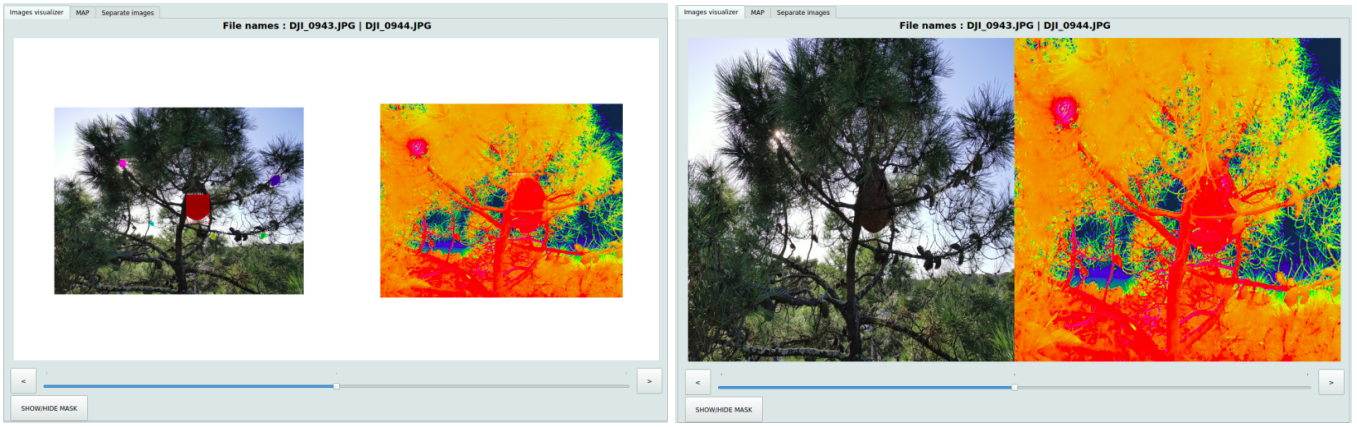

Fig 10: Bouton SHOW/HIDE MASK, Avant / Après

Enfin, nous avons rajouté une carte permettant d'afficher la position de tous les nids localisés par le logiciel avec des points rouge pour les images thermiques et bleu pour les images visibles. Pour éviter d'être noyé par de trop nombreux "faux positifs", on peut augmenter ou diminuer le niveau de tolérance, pour afficher plus ou moins de points sur la carte. La carte peut ainsi être utilisée pour mettre en évidence les endroits suspectés d'être des nids par plusieurs images visibles ET thermiques [Fig 11].

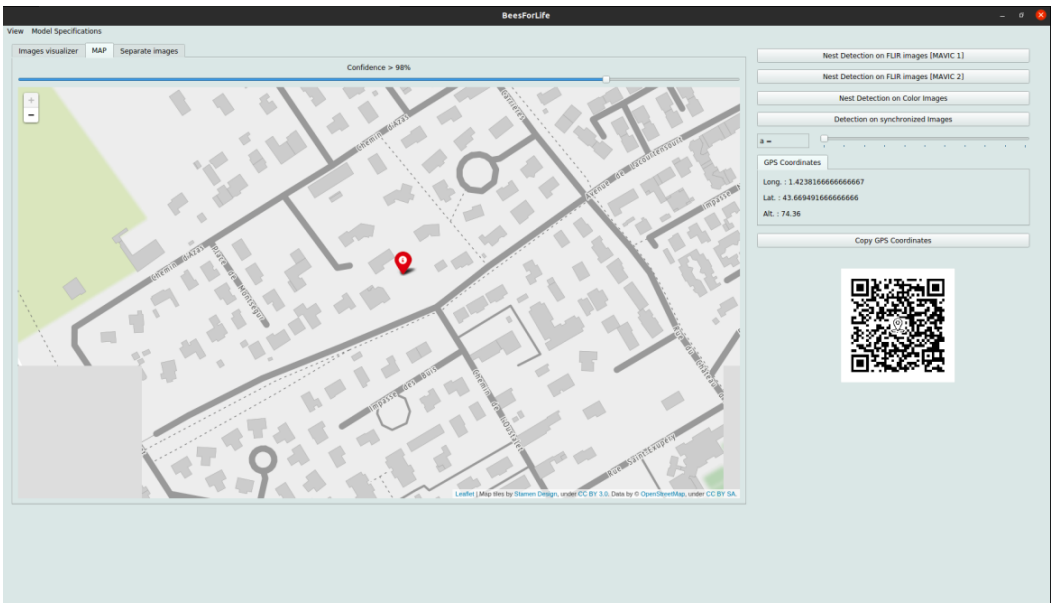

Fig 11: Illustration de la carte avec un nid FLIR localisé.

En complément de cette carte, nous affichons dans le panneau de droite les coordonnées gps de l'image affiché avec un bouton pour copier/coller ces coordonnées rapidement (afin de les retrouver sur *google maps*) ainsi qu'un *QR Code* pour retrouver l'emplacement sur *google maps* directement sur son smartphone ou sa tablette [Fig 12].

Fig 12: Affichage GPS et QRcode.

voici enfin l'affichage final de notre nouvelle interface graphique avec tous les ajouts précédemment mentionnés en [Fig 13].

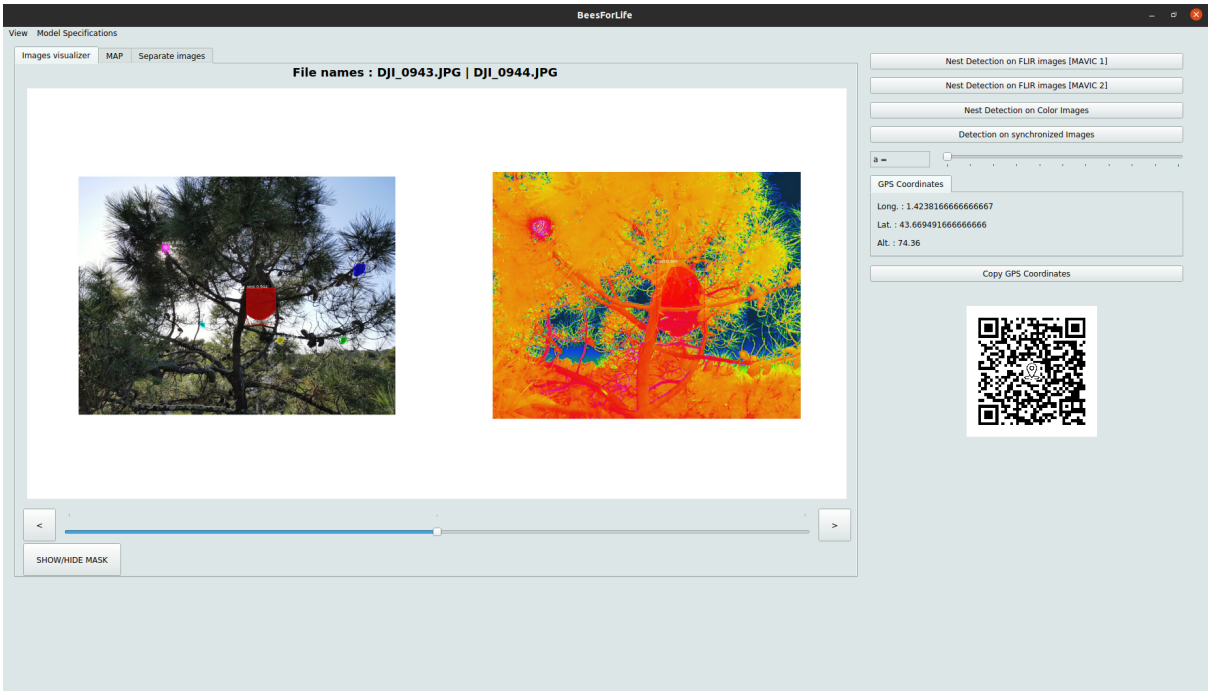

Fig 13: Nouvelle interface graphique.

Vision Transformers

Une autre partie de notre travail consistait à améliorer le système de reconnaissance des nids. Après avoir étudié l'état de l'art nous en avions conclu que les *Vision Transformers* étaient une piste à travailler en tant que principal concurrent des *Masks-RCNN*.Nous nous sommes donc penchés sur une implémentation à base de *ViT*.

La principale difficulté a été de mettre en place une implémentation d'un entraînement *ViT*. Les *ViT* sont relativement récents, les différents travaux et exemples de code sur le sujet datent de 2021 ou 2020 au plus tôt. Les sources sont donc relativement peu nombreuses. S' il a été facile de faire fonctionner un premier exemple montrant l'efficacité des *ViT* sur *CIFAR-10*, il a été plus compliqué de monter notre propre entraînement pour notre projet. C'est pour cela que cette tâche a pris beaucoup plus de temps que prévu: près de 2 semaines pour faire tourner notre premier entraînement.

Nous avons fini par trouver une implémentation du modèle *ViT* utilisable pour commencer nos entraînements. Après pas mal de tests, on a décidé d'utiliser les Machines de Google Colab, ce qui simplifie grandement l'accès aux différentes dépendances. On dispose ainsi également de *TPU* de 12Go pour faire tourner notre entrainement. Ce dernier point est très important puisque l'entraînement demande énormément de mémoire pour fonctionner. De ce fait, nous avons été obligés de diviser par 10 la taille de nos images d'origine pour pouvoir exécuter le code, faute de *RAM*. En effet les images d'origine de 8000x6000 sont inutilisables pour les *ViT* car bien trop grosses pour être exploitables. Nous avons donc été obligés de dégrader drastiquement leur taille pour pouvoir lancer l'entraînement.

Une fois le modèle paramétré nous avons donc pu lancer l'entraînement, pour cela on a utilisé 307 images provenant du nouveau drone : 70% d'entre elles pour l'entraînement et

30% pour le test. Une fois les *numpy arrays (x_train, y_train, x_test et y_test)* remplis on lance l'entraînement.

Notre modèle identifie 2 classes :

- la **classe 0** pour une image de fond,
- la **classe 1** pour une image contenant un nid de frelon.

Résultats des ViT :
sur 307 images :
x_train : 202 images
x_test : 105 images

Après cet entraînement on constate que le *ViT* semble ne pas savoir identifier les nids, il renvoie donc toujours les mêmes probabilités indiquant que nos images sont toutes des nids pour maximiser son résultat, puisqu'il y a plus de nids que d'images sans nids sur nos échantillons. Les résultats sont pourtant très prometteurs sur x train avec très peu d'erreurs, en revanche sur l'échantillon de test on observe le phénomène décrit précédemment.

```
True Positif = 83
True Negatif = 12
False Positif = 3
False Negatif = 2
Success / False : 0.95
```

```
True Positif = 85
True Negatif = 0
False Positif = 15
False Negatif = 0
Success / False : 0.85
```

Fig 14: Résultats observés sur x_train et x_test

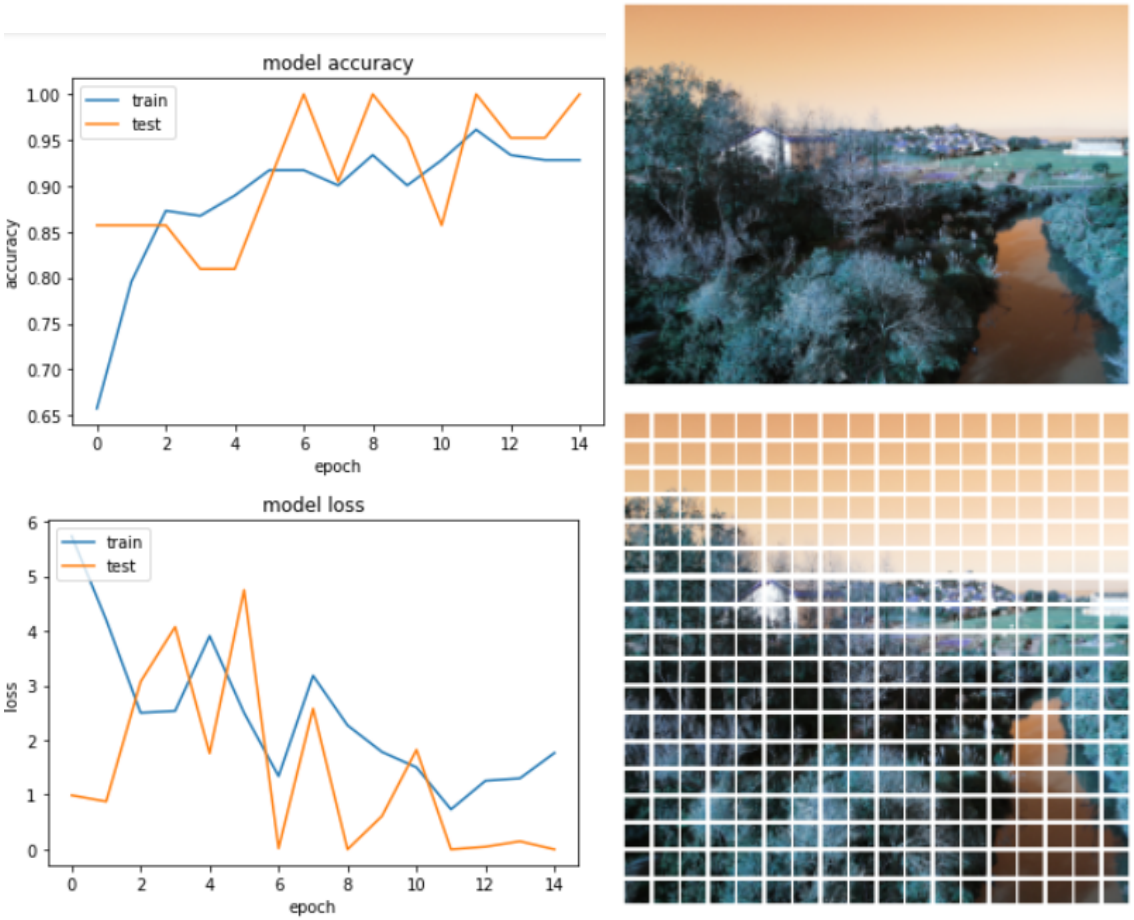

Fig 14: Évolution de la loss et de l'accuracy et découpage de l'image en patchs.

On en conclut que le modèle *ViT* n'est pas utilisable pour notre cas d'utilisation. Le *VIT* est plus efficace pour identifier des objets en plein écran plutôt que de rechercher un petit objet sur une grande image. Il est en effet conçu comme on l'a vu pour chercher des corrélations entre des successions de petits patch. Hors le plus souvent les nids ne sont présents que sur 1 ou 2 patchs, il y a donc très peu d'informations utiles sur les successions d'images et il est difficile pour le logiciel d'identifier les nids de frelons par cette méthode. Nous n'avons donc pas retenu cette solution et nous sommes finalement concentrés sur l'amélioration des *Masks-RCNN*.

Mask-RCNN et Transfer Learning

Les *Masks-RCNN* semblent être la meilleure méthode pour localiser les nids de frelons, nous nous sommes donc concentrés sur cette technique pour continuer notre travail et répondre à la tâche 1.2 :

> **tâche 1.2.** Mettre à jour l'entraînement de l'application pour le faire fonctionner au mieux sur les images du nouveau drone.

Nous avons donc traité toutes les images à disposition du nouveau drone pour ré-entraîner notre réseau de RCNN en utilisant le *Transfer Learning.*

La première étape a été d'annoter tous les nids présents sur les nouvelles images du drone DJI à l'aide d'un outil en ligne permettant de générer un fichier *.json* avec les coordonnées des polygones englobants les nids [Fig 15].

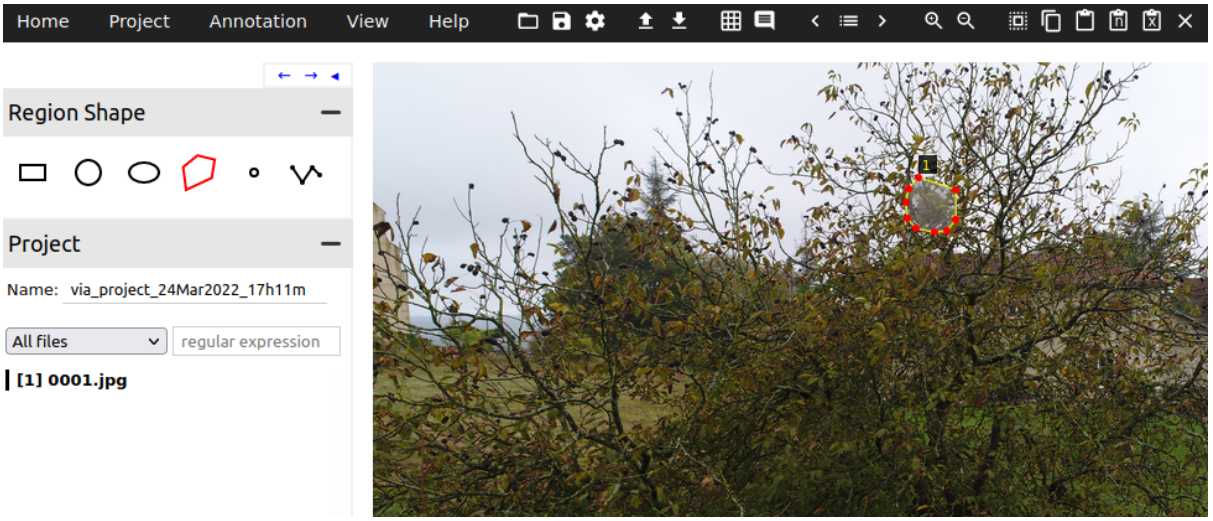

Fig 15: Exemple d'annotation
(source: https://www.robots.ox.ac.uk/~vgg/software/via/ [5])

La difficulté pour entraîner sur ces nouvelles images étant la taille de celles-ci : 8000x6000 pixels : soit 32M pixels. L'image est donc très grande, ce qui rend les entraînements très coûteux en temps et en mémoire vive, comme on l'a vu pour les *ViT*. Nous avons donc ici aussi décidé de réduire nos images en dimension par un facteur 4, les passants ainsi à une

taille de 2000x1500 = 3M pixels soit plus de 10x moins. Nous avons alors un dataset de 235 images labellisés réparties en 100 images dans train et 100 images dans val et 35 dans test.

Nous avons eu quelques difficultés de compatibilités pour entraîner le réseau qui tourne sous une vieille version de *tensorflow* : la 1.8.0 à cause de la librairie *mrcnn*. Nous avons finalement réussi à la faire fonctionner sous *tensorflow* 1.13.1 sur Google Colab pour bénéficier d'une bonne puissance de calcul : 12Go de RAM sur Carte Graphique afin de travailler convenablement.

Après plusieurs tests, nous avons concilié temps de calculs et résultats en entraînant notre réseau durant 50 epochs de 30 étapes. Pour atteindre au final une loss de 0.1027.

```
30/30 [==============================] - 142s - loss: 0.1741
Epoch 34/50
30/30 [==============================] - 141s - loss: 0.1594
Epoch 35/50
30/30 [==============================] - 141s - loss: 0.1412
Epoch 36/50
30/30 [==============================] - 141s - loss: 0.1600
Epoch 37/50
30/30 [==============================] - 140s - loss: 0.1723
Epoch 38/50
30/30 [==============================] - 140s - loss: 0.1434
Epoch 39/50
30/30 [==============================] - 141s - loss: 0.1739
Epoch 40/50
30/30 [==============================] - 141s - loss: 0.1536
Epoch 41/50
30/30 [==============================] - 141s - loss: 0.1347
Epoch 42/50
30/30 [==============================] - 140s - loss: 0.1331
Epoch 43/50
30/30 [==============================] - 140s - loss: 0.1282
Epoch 44/50
30/30 [==============================] - 141s - loss: 0.1333
Epoch 45/50
30/30 [==============================] - 141s - loss: 0.1102
Epoch 46/50
30/30 [==============================] - 141s - loss: 0.1273
Epoch 47/50
30/30 [==============================] - 141s - loss: 0.1359
Epoch 48/50
30/30 [==============================] - 141s - loss: 0.1278
Epoch 49/50
30/30 [==============================] - 142s - loss: 0.1027
Epoch 50/50
30/30 [==============================] - 142s - loss: 0.1288
```

Fig 16: Déroulement des epochs du nouveau modèle MRCNN couleurs.

Nous avons répété la méthode sur un dataset d'images thermique afin d'obtenir de nouveaux poids pour les nouvelles images thermiques du drone. Voici ci-dessous un récapitulatif de l'évolution de la *loss* sur l'entraînement de nos deux modèles thermiques et visibles [Fig 17].

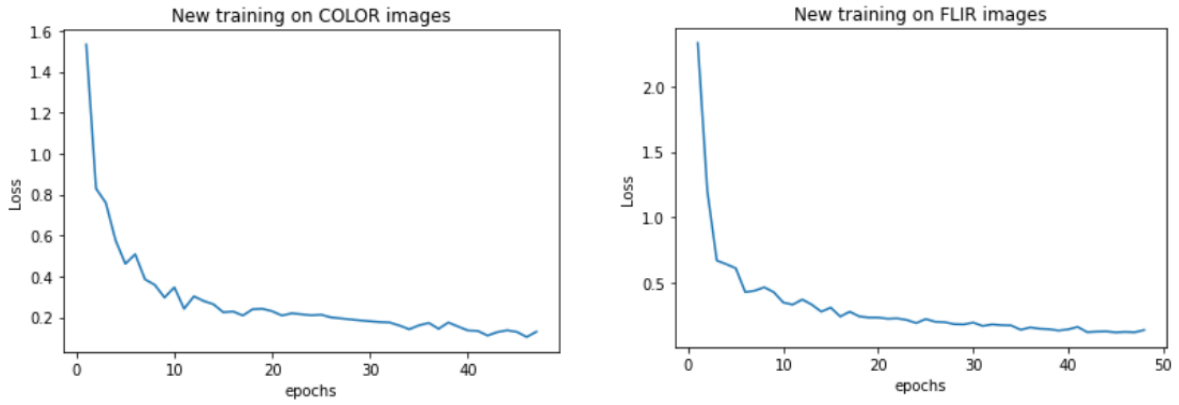

Fig 17: Évolution de la loss pour les modèles visible et thermique.

Une fois l'entraînement terminé, nous avons pu mettre à l'épreuve nos nouveaux modèles pour les comparer aux anciens et évaluer leurs performances.

Analyse des Résultats

La difficulté pour évaluer les performances des nouveaux modèles est que nous disposons d'assez peu d'images du nouveau drone. Notamment nous n'avions pas de jeu d'images correspondant à un vol typique complet, les tests sont donc concentrés sur des extraits d'images de plusieurs vols montrant des nids et d'autres images n'en contenant pas.

Nous avons formé 4 jeux de 100 images pour comparer les modèles sur un pied d'égalité :
100 images visibles du nouveau drone.
100 images thermiques du nouveau drone.
100 images visibles de l'ancien drone.
100 images thermiques de l'ancien drone.

Sur chacun de ces jeux nous avons comparé les résultats obtenus avec l'ancien et le nouveau modèle, voici les résultats obtenus dans [Fig 18] et [Fig 19].

légende :
TP = *True Positif* : Nid de frelon correctement trouvé.
FP = *False Positif* : Objet quelconque mal identifié comme étant un nid de frelon
FN = *False Negatif* : Nid de frelon qui n'a pas été trouvé par le système.

IMAGES VISIBLE	
le dataset : 100 images du **nouveau** drone 74 images sur les 100 contiennent un nid.	le dataset : 100 images de **l'ancien** drone 33 images sur les 100 contiennent un nid.
Nouveaux Poids: TP : 65 FP : 45 FN : 9 88% des nids détectés 59.1% de vrai nids parmis les détections	**Nouveaux Poids:** TP : 29 FP :48 FN :4- 87.9% des nids détectés 37.7% de vrai nids parmis les détections
Anciens Poids: TP : 35 FP : 63 FN : 39 47% des nids détectés 35% de vrai nids parmis les détections	**Anciens Poids:** TP : 9 FP : 63 FN : 24 27.3% des nids détectés 12.5% de vrai nids parmis les détections

Fig 18: Résultats des tests entre l'ancien modèle et le nouveau modèle

On en conclut que dans le domaine visible, le nouvel entrainement à drastiquement amélioré les performances de notre réseau : puisque l'on trouve 2x plus de nids sur les images du nouveau drone et, plus surprenant, presque 3x plus sur les anciennes images.

On remarque aussi qu'il y a moins de Faux positifs, ce qui signifie un gain de temps pour l'utilisateur qui aura moins de faux signalement à vérifier.

IMAGES THERMIQUES	
le dataset : 100 images du **nouveau** drone 53 images sur les 100 contiennent un nid.	le dataset : 100 images de **l'ancien** drone 32 images sur les 100 contiennent un nid.
Nouveaux Poids: TP : 46 FP : 17 FN : 7 86.8% des nids détectés 73% de vrai nids parmis les détections	**Nouveaux Poids:** TP : 8 FP : 10 FN : 24 33.3% des nids détectés 44.4% de vrai nids parmis les détections
Anciens Poids: TP : 20 FP : 176 FN : 33 37.7% des nids détectés 10.2% de vrai nids parmis les détections	**Anciens Poids:** TP : 27 FP : 66 FN : 5 81.8% des nids détectés 29% de vrai nids parmis les détections

Fig 19: Résultats des tests entre l'ancien modèle et le nouveau modèle

On constate ici des résultats très différents, les images thermiques sont très différentes selon la source, le nouveau modèle est donc à utiliser sur les nouvelles images où il trouve 85.8% des nids contre 37.7%, mais il faut garder l'ancien modèle pour les anciennes images où il prévaut avec 81.8% de nids trouvés contre 33.3%.
Ces résultats peuvent s'expliquer par la grande différence entre les précédentes images thermiques et les nouvelles, notament sur la gamme de couleur, les critères d'identification du réseau sont donc bien différents [Fig 20].

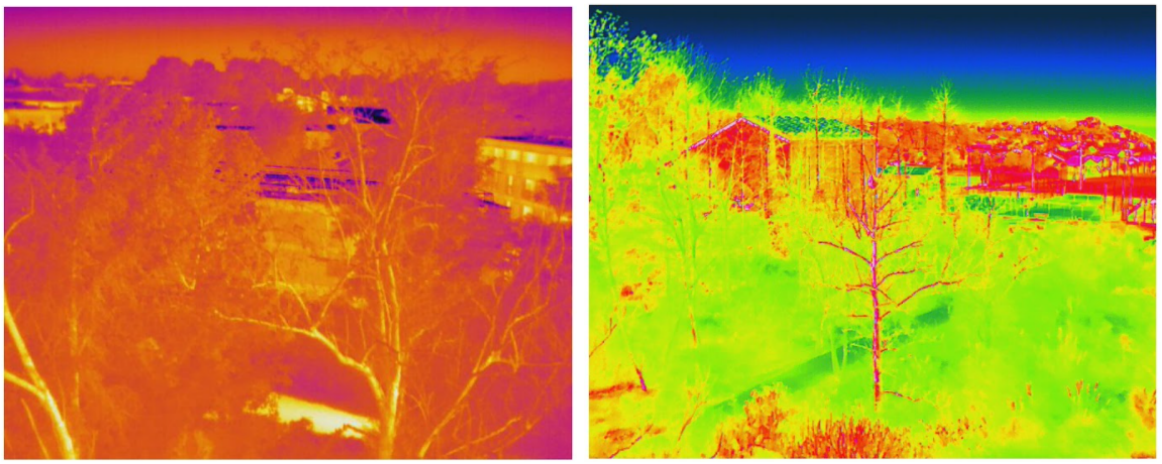

Fig 20 : Exemple d'images thermique de l'ancienne caméra (gauche) et de la nouvelle (droite)

En définitif, l'entraînement que nous avons réalisé sur les nouvelles images DJI à permis d'améliorer efficacement le système, surtout sur les images du nouveau drone. Il restera au client à tester ces résultats théorique sur le terrain dès lors que la nouvelle saison des

frelons asiatiques aura commencé, ce n'est qu'à ce moment que nous pourront être fixés sur les améliorations apportées au logiciel.

Tâches abandonnées

Certains objectifs n'ont malgré tout pas pu être tenu et certaines fonctionnalités ont du être abandonné pour différentes raisons :

tâche 5.1. Analyser l'image thermique pour en extraire la température du nid.

Cette tâche secondaire a dû être abandonnée par manque d'informations, nous avons passé un certain temps à étudier le fonctionnement des images thermiques mais nous ne sommes pas parvenus à extraire les informations voulues de nos images. En effet ces derniers sont censés contenir des métadonnées pour nous permettre d'échelonner les températures à partir des valeurs lus mais nous n'avons pas pu mettre la main dessus malgré de nombreux essais. Etant donné le manque de temps et le fait que cette tâche soit considérée comme secondaire par notre client, nous avons décidé de l'abandonner.

tâche 3.4. Calculer la position précise d'un point sur une image.

Pour cette tâche, le client souhaitait localiser plus précisément les nids sur les images. Au lieu d'utiliser uniquement la position du drone, il était question de localiser le nid sur l'image avec précision. Après un début de recherche, les calculs se sont avérés plus compliqués que prévu à mettre en place. Nous avons tenté de l'ajouter juste avant la fin mais nous avons dû abandonner par manque de temps pour ne pas rajouter de nouveaux bugs juste avant la date de rendu. Il aurait fallu calculer l'altitude du drone par rapport au sommet des arbres, estimer l'angle de la caméra, l'orientation du drone par rapport à sa position précédente et enfin convertir le décalage du nid par rapport au drone en coordonnées gps. Toutes ces étapes étaient également soumises à de lourdes approximations, le résultat risquait de ne pas être à la hauteur du travail en termes de précision.

tâche 6.1. Transformer l'application en site internet pour la rendre accessible à tous.

Enfin, pour cette dernière tâche, le client avait pour projet de créer une plateforme web pour permettre à d'autres personnes de déposer leurs images à faire analyser par notre logiciel moyennant finance. Cette tâche bien plus importante que les autres pourrait faire l'objet d'un PFE à elle seule, nous n'avons pas eu le temps de la réaliser comme nous nous y attendions depuis le début et d'un commun accord avec le client, nous l'avons rapidement mise de côté.

Bees for Life fera probablement l'objet de futurs PFE ou de stages à venir, et nous espérons que les fonctionnalités que nous n'avons pas pu implémenter le seront à l'avenir pour assister notre client.

Conclusion

Travailler sur ce projet nous a beaucoup appris sur plusieurs aspects. Tout d'abord d'un point de vue pédagogique, nous avons beaucoup appris sur le *deep learning*, notamment sur les transformers et leur application au traitement d'image (les *ViT*) que nous ne connaissions pas du tout. Mais également sur la gestion de projet, en fin de formation de master, il est toujours bon d'avoir un rappel du fonctionnement d'un projet sur plusieurs mois avant de nous plonger dans notre stage de fin d'année et la vie active dans notre cas. Et enfin sur le domaine des frelons asiatiques cette fois, plus loin du cadre du développement nous avons pu apprendre beaucoup de choses sur le mode de vie de ces insectes et la menace qu'ils représentent pour notre environnement.

Ce projet de fin d'année a donc avant tout était une énorme plus value pour nous en termes d'expérience. Mais nous avons aussi pu aider Lionel Willaert et son association *Bees for Life* dans leur travail. Nous espérons que les différentes améliorations apportées au logiciel porteront leur fruit et qu'ils pourront travailler plus efficacement en évitant les pertes de temps.

Nos principaux objectifs ont été accomplis, l'algorithme de *deep learning* responsable de localiser des nids de frelons a été amélioré d'après nos tests et devrait être plus fiable à l'avenir. Le logiciel est également beaucoup plus facile d'utilisation et offre de nombreuses nouvelles fonctionnalités pour simplifier l'accès aux informations : notamment l'affichage simultané des images visibles et thermiques ou la géolocalisation des résultats sur une carte, autant d'améliorations qui devraient faire gagner beaucoup de temps aux utilisateurs.

En définitif, nous sommes plutôt satisfait du travail accompli lors de ce projet de fin d'étude, même si nous avons perdu beaucoup de temps sur certaines tâches que nous pensions plus simple, notamment sur la mise en place du système *ViT* qui n'a finalement pas apporté de résultats viables. On aurait ainsi aimé pouvoir commencer à travailler sur l'élaboration d'une plateforme web permettant d'utiliser le logiciel. Cela devra donc faire l'objet d'un autre projet à la charge d'autres étudiants ou stagiaires à l'avenir.

Références

[1] T. Shams and P. Desbarats. Detection of asian hornet's nest on drone acquired flir and color images using deep learning methods. *In 2020 Tenth International Conference on Image Processing Theory, Tools and Applications (IPTA)*, pages 1–6, 2020.

[2] Kaiming He and Georgia Gkioxari and Piotr Dollár and Ross B. Girshick, Mask R-CNN, *CoRR*, 2017

[3] Ashish Vaswani and Noam Shazeer and Niki Parmar and Jakob Uszkoreit and Llion Jones and Aidan N. Gomez and Lukasz Kaiser and Illia Polosukhin, Attention Is All You Need, *CoRR*, 2017

[4] Alexey Dosovitskiy and Lucas Beyer and Alexander Kolesnikov and Dirk Weissenborn and Xiaohua Zhai and Thomas Unterthiner and Mostafa Dehghani and Matthias Minderer and Georg Heigold and Sylvain Gelly and Jakob Uszkoreit and Neil Houlsby, An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, *arXiv:2010.11929*, June 2020

[5] Abhishek Dutta and Andrew Zisserman. 2019. The VIA Annotation Software for Images, Audio and Video. In Proceedings of the 27th ACM International Conference on Multimedia (MM '19), October 21–25, 2019, Nice, France. ACM, New York, NY, USA, 4 pages.